AREA AND SPAN BASED Z-BUFFER

Konstantine I. Iourcha, Roger Swanson and Axel Schildan

CROSS REFERENCE TO RELATED APPLICATION

[0001] The present application claims the benefit of U.S. Provisional Patent Application No. 60/193,736 filed on March 30, 2000 entitled "Span Based Z-Buffer", the subject matter of which is herein incorporated by reference.

BACKGROUND OF THE INVENTION

1. Technical Field

[0002] The present invention relates generally to computerized three-dimensional (3D) graphics image processing, and more particularly to techniques for Hidden Surface Removal (HSR), otherwise known as Visible Surface Determination (VSD), in 3D graphics processing.

2. Description of the Background Art

[0003] Rendering realistic 3D graphical images for display on a two-dimensional computer screen requires processing of information pertaining to relative depth of particular objects within an image. Typically, Hidden Surface Removal (HSR) (i.e., Visible Surface Determination (VSD)) is utilized because objects, or portions of such objects, which are hidden behind other objects should not be displayed on the two-dimensional computer screen.

[0004] One common HSR technique is the use of a Z-buffer containing depth information (Z-values), which indicates a relative depth for each pixel of the computer screen. The Z-values, which can typically comprise 16 or 32-bits per pixel, are stored by the 3D graphics processing engine in a memory, along with color information regarding each pixel, for manipulation and display on the computer screen.

[0005] However, the use of Z-values dramatically increases the memory storage requirement of a 3D graphics processing subsystem. For example, if color information for each pixel comprises 16-bits and the Z-value for each pixel comprises 16-bits, then storage of the Z-value requires a doubling in memory capacity. Use of Z-values having 32-bits for 16 bit per pixel images requires tripling of memory capacity. Accompanying the dramatic increase in memory size is the increase in the amount of data that must be transferred between the graphics processor and the memory. Many graphics operations which require manipulation of the Z-buffer require two to three times more data transfers between the graphics controller and the memory. As resolution increases in the X and Y directions, the required memory accesses grow linearly with their product.

[0006] Therefore, there is a need for a system and method which reduces the storage and bandwidth requirements for implementing HSR/VSD in 3D graphic image processing.

09823163.033001

SUMMARY

[0007] The present invention overcomes or substantially alleviates prior problems associated with storage and bandwidth requirements for implementing HSR/VSD in 3D graphic image processing. In general, the present invention decreases the amount of data required to represent depth information for 3D images. In accordance with one embodiment of the present invention, depth information is represented by a piecewise function Z(x,y). An (x,y) space is split into areas representing a region of primitive, polygonal, or other shapes. For each of these regions, Z(x,y) is defined as a simple parametric analytical function. Subsequently, only a few parameters are required to encode this function in each region. In a similar fashion, an analytical representation may be used to define the area split.

[0008] By using these parametric analytical functions to represent depth value of the span or Z-area split, the present invention achieves several advantages. The first advantage is that the required storage space decreases in practical cases. Secondly, the required bandwidth typically decreases while the processing speed increases. This occurs because in each (x,y) area of the split, the visibility of a primitive being rendered can be estimated analytically with a fixed response time regardless of the number of pixels in the region. Furthermore, the piecewise function may be linear or non-linear.

[0009] In one embodiment, the (x,y) space may also be divided into span segments where a span may be the intersection of the primitive object and a scanline. This may be done to simplify the hardware implementation. Upon receiving the spans for a triangle or other types of polygons or shapes, a module in one embodiment of the invention performs visible surface determination by comparing depth information for each span of the triangle with depth information defined by Z functions of the corresponding segments (segments which this particular primitive span overlaps) stored in the span Z-buffer.

[0010] Storing depth information for each segment of a scanline can dramatically decrease the amount of memory required for storage of depth information because each segment typically includes several pixels. Accompanying this decrease in storage requirement is a decrease in memory bandwidth required to store and retrieve such depth information. Moreover, consolidation of depth values for pixels that are

part of the same object in the same scanline reduces the number of depth value comparisons required in HSR/VSD. Comparisons with individual depth values are minimized and replaced with comparisons of depth values for groups of pixels represented by segments (sub-spans).

[0011] Further embodiments employing the principles of the present invention may take advantage of the coherence in adjacent pixels belonging to the same drawing primitive. This enables the detection of the visibility of several pixels at a time and reduces the number of required memory accesses. For a given Z-buffer size, embodiments employing the principles of the present invention can achieve a higher depth resolution than a standard Z-buffer with the same number of bits. Advantageously, the required memory accesses grow linearly with resolution in the Y-direction only.

09823463.033001

BRIEF DESCRIPTION OF THE DRAWINGS

[0012] FIG. 1 is a block diagram of a computer system for implementing the present invention;

[0013] FIG. 2A is a block diagram illustrating the operations of a 3D graphics engine;

[0014] FIG. 2B is a block diagram illustrating alternative operations of a 3D graphics engine;

[0015] FIG. 3A is a viewport showing a piecewise function Z(x,y);

[0016] FIG. 3B is an alternative embodiment wherein the viewport area is divided into spans;

[0017] FIG. 3C is an illustration of how depth information is stored as a linear function for each segment.

[0018] FIG. 3D is an illustration of the updating of a scanline to remove hidden objects;

[0019] FIG. 4 is a flowchart illustrating one embodiment of the present invention;

[0020] FIG. 5 is a block diagram of an exemplary data structure for storage of span segments;

[0021] FIG. 6A is a block diagram of another example of a data structure for storage of span segments; and

[0022] FIG. 6B. is a specific embodiment of the data structure of FIG. 6A.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

[0023] FIG 1 is a block diagram illustrating a computer system 100 capable of implementing the present invention. As shown, the computer system 100 includes at least one input device 102, a display device 104, a CPU 106, memory 108 and a graphics engine 110. The input device 102 may include a keyboard, mouse or similar device for the input of information to the system 100. The display device 104 may be a monitor, television or similar device which shows objects generated by the present invention. The CPU 106 typically performs a majority of the computational operations of the computer system 100 except for those related to the display device 104. Optionally, CPU 106 may perform the computational operations related to the display device 104.

[0024] The graphics engine 110 preferably includes a graphics coprocessor which performs the calculations necessary for high quality, high speed graphics required in 3D applications. According to one embodiment, the graphics engine 110 may include a frame buffer having a Z-buffer.

[0025] In an embodiment, the graphics engine 110 may also include a 3D graphics engine 200 as shown in FIG. 2A. The 3D graphics engine 200 forms a portion of a 3D image rendering pipeline (not shown) that generates 3D image data in the form of a plurality of primitive objects 202. Although the primitive object 202 is shown as a triangle, those skilled in the art will realize that other 3D primitive objects such as lines, quadrilaterals and polygons may be contemplated. The remaining portions of the 3D image rendering pipeline are preferably conventional elements omitted in the drawings to assist in describing the functionality of the present invention.

[0026] The 3D graphics engine 200 receives information about primitive object 202 in the form of 3D positional coordinates. As shown in FIG. 2A, the exemplary primitive object 202 is a triangle. Thus, the 3D positional coordinates include each of the three vertices of the triangle. The 3D graphics engine 200 may encompass a variety of other functions to render realistic 3D-type graphical images on a two-dimensional computer display. Preferably, the 3D graphics engine 200 also receives rendering parameters to generate any one of a plurality of 3D effects including fogging, shading, texturing and lighting. Further functions of a 3D graphics engine are

described in Foley, et al., Computer Graphics: Principles and Practices, Second Edition in C, Addison-Wesley Publishing Company (1990).

[0027] In a preferred embodiment of the present invention, the 3D graphics engine 200 includes a span generator 204 and a visible surface determination module 206. The span generator 204 receives 3D coordinates for each of the vertices of primitive object 202. Based on the coordinates, the span generator 204 then generates spans for the primitive object 202. The span data is then forwarded to the visible surface determination module 206 where depth information for each span is compared with depth information defined by an area represented by a piecewise function Z(x,y) of primitive object 202 and/or a comparison with at least one additional object to determine the visible surface. Piecewise function Z(x,y) will be discussed in more detail in connection to FIGs. 3A-3D. The comparison process includes the generation of span segments (sub-spans) based on intersection points with other spans, with each span segment being indicative of a visible portion of a span of a particular primitive object 102. The output of the visible surface determination module 206, which identifies the span segments, is then preferably stored in a storage device 208. Optionally, the output of the visible surface determination module 206 may be stored in storage device 108 (FIG. 1).

[0028] In one embodiment of the present invention, depth information is represented by a piecewise function Z(x,y) as shown in FIG. 3A. An (x,y) space, defined by a window or viewport 300, is split into regions representing a region of a primitive, polygon, or other shape. For each region, $Z_1(x,y)$, $Z_2(x,y)$, ... $Z_i(x,y)$ is defined as some simple parametric analytical function, which may be linear or non-linear. Thus, only a few parameters are required to encode this function. Similarly, analytical representations may be used to define splits (e.g., splits 302, 304 and 306). By using analytical functions to represent this data, required storage space and bandwidth typically decreases while the processing speed increases. The processing speed increase is attributed to the fact that for each (x,y) area of the split, the visibility of a primitive being rendered can be estimated analytically with a fixed response time regardless of the number of pixels in the region.

[0029] In an alternative embodiment, the (x,y) space (defined by viewport 320) may be divided into spans to simplify the hardware implementation, as shown in

FIG. 3B. With this embodiment, the visible surface determination module 206 (FIG. 2) is able to compare depth information for each span of the primitive object with depth information defined by Z functions of the corresponding segments (segments which the primitive object span overlaps) stored in the span Z-buffer.

[0030] The viewport 320 is defined by $X_{min} \ldots X_{max}$ and $Y_{min} \ldots Y_{max}$, which are all integer values. Each scanline defined by an integer, y, between Y_{min} and Y_{max} inclusive, and the scanline is split into an integral number of segments, n_y. Parameter n_y may be variable for each scanline and should be stored per scanline. The split is defined by split points $X_{y,i}$, wherein $i = 1 \ldots n_y$ such that $X_{min} = X_{y,1} < X_{y,2} < \ldots < X_{y,n_y} X_{y,ny} < X_{max}$, with the assumption that $X_{y,n_y+1} X_{y,ny+1} = X_{max}$.

[0031] A split is a set of segments $S_{y,i} = [X_{y,i}, X_{y,(i+1)}]$ where $i = 1, 2, \ldots n_y$. For each segment of the split, $S_{y,i}$, $i = 1, 2, \ldots n_y$, there is a separate linear function $Z_{y,i}(x) = a_{y,i} \bullet x + b_{y,i}$, defining depth information on this particular segment. Hence, for each segment $S_{y,i} = [X_{y,i}, X_{y,(i+1)}]$, $i = 1, 2, \ldots n_y$, the coefficients $a_{y,i}$ and $b_{y,i}$ may take on any real value and should be stored in the span Z-buffer structure. Segments $S_{y,i}$ may or may not be equal in length.

[0032] As an example, suppose that for some scanline y, $n_y=5$ and segment end points are $X_{y,1}, \ldots X_{y,5}$. FIG. 3B shows scanline segments $S_{y,1}, S_{y,2}, \ldots, S_{y,5}$, each having an associated function $Z_{y,i}(x)$ (not shown). Thus, $S_{y,1}$ has the function $Z_{y,1}(x)$, while $S_{y,2}$ has the function $Z_{y,2}(x)$, and so forth. Alternatively, in addition to a linear function, $Z_{y,i}(x)$ may take on a non-linear function such as $Z_{y,i}(x) = a1_{y,i}(x^k) + a2_{y,i}(x^{(k-1)}) + \ldots + ak_{y,i}(x) + b_{y,i}$; where k and a1 ... ak may be any real value.

[0033] FIG. 3C illustrates how the depth information is defined by the piecewise function of FIG. 3B. The z-axis represents depth values for which different graphs of the Z functions may exist. The example of FIG. 3C shows the Z functions as $Z_{y,1}(x), \ldots, Z_{y,5}(x)$. Every primitive object is processed scanline-by-scanline (or span-by-span), where the span is the intersection of the primitive object and a scanline.

[0034] FIG. 3D illustrates the updating of a scanline 350 of a span based Z-buffer. According to this embodiment of the present invention, all segments of the scanline 350 which overlap with a primitive object span 352 are determined and processed in an increasing X order (left to right). For each such segment 351 which

overlaps the primitive object span 352, two linear depth functions are defined on intersection 354 of the segment and primitive object span. One linear depth function represents the current depth stored in the Z-buffer. The other linear depth function represents the depth of the primitive object.

[0035] As known by those skilled in the art, by solving a simple linear equation, the visible regions of the primitive object in this segment can be determined (i.e., determine the X coordinate of the intersection point of the two Z functions and compare it with the end points of the segments). For a particular Z-buffer segment and primitive object span, there may be at most two different visibility segments with one visibility segment corresponding to a non-visible portion of the primitive object and the other segment corresponding to a visible portion of the primitive object.

[0036] Once visibility is determined, the Z-buffer structure must be updated. The Z function will change only in the visible segment of the primitive object. In the worse case, the original Z-buffer segment may be split into three segments – two of which will keep the original segment Z function from the Z-buffer and one in which a new Z function is assigned. Specifically, the new Z function will be the Z function of the visible portions of the primitive object.

[0037] After the first segment of the Z-buffer is processed, the next segment is then processed in a similar way. For some of the neighboring new segments (coming from different segments of the Z-buffer), the same primitive object Z function may be obtained if the primitive object is visible in both segments. In this situation, the neighboring segments can be merged into one segment which would prevent excessive granularity of the span Z-buffer splits and save memory space.

[0038] Thus, the embodiment of FIG. 3C-D performs processing on a scanline-by-scanline manner. After all spans of the primitive object have been processed, a next primitive object may be processed in the same manner, if one exists.

[0039] FIG. 4 is a flowchart 400 illustrating a preferred method for determining span segments according to the present invention. In block 402, the span generator 204 (FIG. 2) generates spans for a primitive object. The spans are based on 3D coordinates of vertices and other surface variations of the primitive object. The span generator 204 then forwards the span information to the visible surface determination module 206 (FIG. 2) for further processing.

[0040] The visible surface determination module 206 determines if any of the spans are overlapping in block 404. This determination is performed by comparing depth information for each span with depth information defined by an area represented by a piecewise function Z(x,y) of another object or another part of the same object. The various methods for generating the piecewise function Z(x,y) is described in connection with FIGs. 3A-3D.

[0041] If there is an overlap, intersection points of the span with the area represented by the piecewise function Z(x,y) are determined in block 406 by the visible surface determination module 206. Based on these intersection points, the visible surface determination module 206 generates span segments in block 408. These span segments or sub-spans indicate visible portions of the spans of the particular primitive object.

[0042] Finally, in block 410, the outputs of the visible surface determination module 206 are stored in a storage device such as device 208 (FIG. 2).

[0043] There are numerous ways to store the output of the visible surface determination module 206. FIG. 5 illustrates one data structure for storage of the span segment (sub-span) information. In FIG. 5, the span segment data is stored in a linked list for each Y. Each set of span segment data includes position data (i.e., $x_{y,i}$), depth data coefficients $a_{y,i}$, $b_{y,i}$ (or the equivalent set of non-linear coefficients) and data for the memory location in which data coefficients for the next span segment is stored. As shown in FIG. 5, the data is stored according to scanlines. The position data, therefore, only includes position data along the horizontal (X) axis, with the vertical (Y) axis position data being implicit in the data structure and known by the 3D graphics engine 200. Those skilled in the art will recognize that depth functions may be defined by parameters other than by ($a_{y,i}$, $b_{y,i}$), and can be a non-linear function.

[0044] In FIG. 6A, the data for each scanline 600 is organized in a binary tree structure 602. A binary tree is a set of nodes wherein one node is designated the root and the remaining nodes form, at most, two sub-trees. With the embodiment of FIG. 5, the average time to find a sub-span is one-half of the number of sub-spans. In the embodiment of FIG. 6A, balanced trees may be used to increase access speed. A binary tree is balanced if the height of the left subtree of every node never differs by more than ±1 from the height of its right subtree. Furthermore, the binary tree supports insertion

and deletion of $x_{y,i}$ (used as a key and stored in the nodes of the tree) as well as fast access to the leftmost segment overlapping with a given primitive object span. Thus, the average time to find a sub-span is proportional to the base-2 logarithm of the number of sub-spans in the scanline.

[0045] The binary tree structure of FIG. 6A may store the sub-spans for a scanline as a function of the depth relationship of the sub-span as shown in FIG. 6B. In the example of FIG. 6B, span segment 1 has a greater depth value than span segment 3, and has a smaller depth value than span segment 2. Hence, span segment 1 is behind, and possibly partially or entirely hidden by span segment 3. Span segment 1 is in front of and may partially or completely block span segment 2. Span segment 2 is similarly situated with respect to span segments 4 and 5, while span segment 3 is similarly situated with respect to span segments 6 and 7. Thus, the depth relationship for each span segment is inherent in the data structure of FIG. 6B. As such, a comparison of depth value is not necessary in order to determine the relative depth of two span segments, and processing time is reduced. Those skilled in the art will recognize that other forms of binary trees and other forms of data structures can be used in the implementation of the present invention. Additionally, other high speed access methods may be used according to a particular design need.

[0046] In an embodiment where many span segments comprise only a few pixels, it may be preferable to store an individual depth value for each pixel, such as in a conventional Z-buffer. Additionally, storage of depth information, such as Z-values, is necessary where more than one span segment is designated for storage in the same memory location. In such cases, depth information for the pixels in the scanline covered by the two conflicting span segments is stored as conventional Z-values. Preferably, the 3D graphics engine 200 allows for graceful degradation of span-based information to Z-values for individual pixels. Thus, information for certain scanlines can be stored in the form of span segments as described above, and information for certain other scanlines may be stored in the form of z-values for individual pixels. This process allows for efficient storage and retrieval of depth information. In a preferred embodiment, depth information may be stored in the form of span segments or in the form of z-values for different pixels in the same scanline.

[0047] Preferably, the 3D graphics engine 200 described herein can be implement in hardware contained on one or more integrated circuits. The exact manner in which the functions and features described here are implemented is not critical and can be implemented by use of known hardware structures and design techniques. A hardware implementation is preferable as it provides better performance. However, other considerations such as cost, die size and ease of manufacturing and testing may dictate implementation of certain features and functions with a combination of hardware, software and firmware.

[0048] It is also within the scope of the present invention to implement a program or code that can be stored in an electronically readable medium to permit a computer or similar device to perform any of the methods described herein.

[0049] The invention has been described above with reference to specific embodiments. It will be apparent to those skilled in the art that various modifications may be made and other embodiments can be used without departing from the broader scope of the invention. Therefore, these and other variations upon the specific embodiments are intended to be covered by the present invention, which is limited only by the appended claims.

09823573.030301